Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

exam – **Embedded Software** – TI2726-B
January 28, 2019 13.30 - 15.00

———

This exam (6 pages) consists of 60 True/False questions.
Your score will be computed as: $max(0, \frac{\#correct}{60} - \frac{1}{2}) \times 2 \times 9 + 1$
It is **not** allowed to consult the book, handouts, or any other notes.

Instructions for filling in the answer sheet:

- You may only use a **B-pencil** so erasures can be applied to correct mistakes.
- Fill in the boxes **completely**.
- Answer **all** questions; there is no penalty for guessing.
- Do not forget to fill in your **Name** and **Student Number**

The following abbreviations are assumed to be known:

- RR (Round Robin)
- RRI (Round Robin with Interrupts)
- FQS (Function Queue Scheduling)
- RTOS (Real-Time Operating System)
- ISR (Interrupt Service Routine)
- UART (Universal Asynchronous Receiver Transmitter)

One system clock tick = 10 ms (unless stated otherwise).

We make use of the following definitions:

```
void delay(int ms) {
   !! do some CPU computation to the number of ms milliseconds
}

void putchar(char c) {
   while (!! UART tx buffer not empty)
      ;

   !! send c to UART tx buffer
}

void puts(char *s) {
   !! write string s using putchar
}
```

1. A defining characteristic of embedded systems is the need for large volumes of scale. true/false

2. The Underground Tank Monitoring System is a classic example of an embedded system in that it involves input (sensors/buttons), output (display/printer) and real-time constraints. true/false

3. Because embedded software engages the physical world, it has to embrace time and other non-functional properties, which requires the use of interrupt handlers to guarantee responsiveness. true/false

4. An embedded program can be coded as a finite state machine where all state transitions are triggered by user actions. true/false

5. Several models of computation for embedded systems are described in [Lee:2002].
- The ROS software (used in the practicals) is a prime example of the Dataflow model. true/false

6. An interrupt is an asynchronous signal from hardware to indicate the need for processor attention. true/false

7. Finite State Machines can be coded in VHDL.
- An advantage of doing so is that it results in lower interrupt latency as less context (e.g., registers) need to be saved and restored. true/false

8. Finite State Machines can be coded in a number of ways in C.
- In the function-based solution, transitions (arcs) are encoded as a function calls. true/false

9. Global variables are located on the data heap by the C runtime support at start of execution. true/false

10. The C language is centered around the `int` data type, which is defined to hold integral numbers of at least 16 bits. true/false

11. GDB is programming tool that aids memory debugging by executing a program in a safe environment. true/false

12.

```
int main(void)
{
   int c;
   statefp state = start;
   while((c = getchar()) != EOF) {
      state = (statefp) (*state)(c);
   }
   return 0;
}
```

The above loop drives the FSM until all characters from the standard input have been processed. true/false

13. Specifying the type of `statefp` is difficult in C because it is recursive and types cannot be referenced before being fully defined.
- This explains the need for an explicit type cast in the body of the while loop. true/false

14. Using interrupts improves task response time. true/false

15. An interrupt service routine does not need to be allocated its own call stack. true/false

16. A low-priority ISR can be interrupted by a high-priority task. true/false

17. Since disabling interrupts increases interrupt latency, several alternative methods have been developed for dealing with shared data, including writing so-called "ingenious code".

```
volatile static long int lSecondsToday;
void interrupt vUpdateTime()
{
   ++lSecondsToday;
}
long lGetSeconds()
{
   long lReturn;
   lReturn = lSecondsToday;
   while (lReturn!=lSecondsToday)
      lReturn = lSecondsToday;
   return (lReturn);
}
```

The **volatile** keyword is needed to prevent the compiler from optimizing the loop away. true/false

18. When a processor in an embedded system is powered up, interrupts are enabled to meet response-time requirements. true/false

19. The shared-data problem can be solved by storing data in non-volatile memory. true/false

20. An interrupt vector table contains the addresses of the interrupt service routines. true/false

21. Given the following pseudo code, which reads the current values of 3 different buttons and acts accordingly. The 3 buttons are all mapped to bits 0..2 of the button register. The buttons are already debounced.

```
void f1(void) { delay(1000); }
void f2(void) { delay(2000); }
void f3(void) { delay(3000); }

void main (void) {
   while (1) {
      if (buttons & 0x01) f1();
      delay(1000);
      if (buttons & 0x02 ) f2();
      delay(1000);
      if (buttons & 0x04 ) f3();
   }
}
```

This code is an example of an RR architecture. true/false

22. When none of the buttons have been pressed, the longest time that button 2 must be pressed to activate f2() once is 1 second. true/false

23. When the system is in an arbitrary state, button 1 must be pressed at most 8 seconds to activate f1(). true/false

24. The worst-case latency for servicing an interrupt is a combination of factors, including the longest period of time in which interrupts are disabled. true/false

25. On 8-bit processors the number of interrupt priorities is limited to 256 (2^8). true/false

26. Shared (global) variables marked `static` guarantee atomic access within the code file due to C's data hiding principle. true/false

27. **Priority inversion** occurs when a high priority task blocks on a resource held by a low priority task t that is prevented from running due to some other task(s) with more priority than t. true/false

28. An RRI architecture is most robust to code changes. true/false

29. In an RTOS, tasks can be in state BLOCKED, READY or RUNNING.
- A task can transition directly from BLOCKED to READY. true/false

30. An ISR could activate (unblock) more than one task. true/false

31. A reentrant function may **not** call other functions true/false

32. A queue inbetween a producer and consumer task can be controlled by a counting semaphore that records the number of items in the queue. true/false

33. A program running on an RTOS may create tasks dynamically at runtime.
- the program ends once `main()` and all spawned tasks have finished. true/false

34. Even a local variable can introduce a shared data problem when its address escapes the defining function, for example, by storing the address in a global datastructure. true/false

35. In the implementation of the `OS_Pend()` primitive, the RTOS first switches the state of the current task to BLOCKED, and then looks for a task in the READY queue.
- if the READY queue is empty the processor may be put into sleep mode to save energy when idling. true/false

36. When using an RTOS signaling between ISRs and tasks must be done by calling appropriate RTOS primitives. true/false

37. A function can be made reentrant by temporarily disabling interrupts, but then it may no longer be called by an ISR. true/false

38. The accuracy of a `OSTimeDly()` depends on the frequency of the periodic timer used by the OS.
- the higher the frequency, the lower the accuracy. true/false

39. An RTOS usually provides two types of delay functions: polling-based and timer-based.
- polling-based delays are specified in so-called ticks. true/false

40. The **heartbeat timer** is a single hardware timer an RTOS is using as base for all timings. true/false

41. To address the shared-data problem, many RTOSs provide communication primitives like queues, mailboxes, and pipes.
- the unique property of a mailbox is that it can accept items from different tasks. true/false

42. The advantage of pipes over queues is that messages/items can be of variable length. true/false

43. With the X32 RTOS creating a task amounts to initializing a stack and invoking a context switch to the task's main function.
- This approach provides the possibility to use one stack for multiple (concurrent) tasks and reduce the memory footprint. true/false

44. Consider the following code fragment:

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  extern char *UART_rx_buf;       // copied from <uart.h> for reference
6  extern char *UART_tx_buf;
7  extern char *UART_ier;
8
9  #define LEN 80
10 static char *next_command = NULL;
11
12 void rx_ready() {
13     static char buffer[2][LEN];
14     static int toggle = 0;
15     static char *command = buffer[0];
16     static int cnt = 0;
17
18     char c = *UART_rx_buf;
19     if (c == '\n') {
20         command[cnt] = '\0';
21         next_command = command;
22         toggle = 1 - toggle;
23         command = buffer[toggle];
24         cnt = 0;
25     } else {
26         command[cnt++] = c;
27     }
28 }
29
30 int main() {
31     *UART_ier |= 0x3;           // start RX and TX please
32     while (1) {
33         if (next_command != NULL) {
34             if (strcmp(next_command, "exit") == 0) {
35                 exit(0);
36             } else if (strcmp(next_command, "hello") == 0) {
37                 printf("world\n");
38             }
39             next_command = NULL;
40         }
41         ...
42     }
43 }
```

This code is an example of an RR architecture. true/false

45. Consider lines 5-7 in which some of a UART's registers are declared. This way a UART, or any other peripheral for that matter, can be accessed with normal read/write instructions.
- this mode of operation is called 'memory-mapped I/O'. true/false

46. The function `rx_ready()` uses a technique called 'alternating buffers'.
- the global variable `next_command` signals the `main()` routine which buffer is ready for processing. true/false

47. The code suffers from a (subtle) data sharing bug as both `rx_ready()` and `main()` write to the same global variable `next_command`.
- in certain cases `main()` will read data before `rx_ready()` has written it to the buffer. true/false

48. Removing the write statement on line 39 will not resolve the shared data bug.
- instead `main()` should clear the command by writing a null character to the first position in the buffer (`next_command[0] = 0;`). true/false

49. An alternative approach would be to make use of semaphores to support `rx_ready()` passing the next command to `main()`.
- two semaphores are required; one for signalling and the other for mutual exclusive access to the buffers. true/false

50. Tasks in an RTOS are often structured as state machines with states stored in private variables and ISRs advancing the state machine. true/false

51. In an RTOS each task requires its own stack space. true/false

52. Printing from an ISR is considered bad practice as the driver resides in the RTOS. true/false

53. Time-slicing should be avoided in an RTOS because it introduces the shared-data problem. true/false

54. A semaphore S used by task A must be declared as a local variable within the source code of A. true/false

55. Time slicing between tasks of equal priority is to be avoided as it compromises the predictability of their response times. true/false

56. When developing code for an embedded system, the software can de structured into HW-dependent and HW-independent code.
- Doing so makes debugging HW-independent code feasible on the host platform true/false

57. An in-circuit emulator is preferred to a logic analyzer because it can be used with any type of processor. true/false

58. Although the assert macro is a useful debugging aid during program development, it can only be used on the target machine. true/false

59. A large study of outdoor sensor-network deployments [Beutel:2009] has shown that the two most underestimated problems have been the water-proof packaging of the sensor nodes and the provision of a reliable base station. true/false

60. When debugging code for a distributed sensor network, collecting the (debug) output of the nodes can be arranged in different ways.
- A major advantage of a testbed is that large volumes of (debug) data can be handled. true/false